Chemistry examples

goo.gl/NedF8I

goo.gl/NedF8I

Geant4-DNA tutorial,
November 7, 2014

ESA/ESTEC

Mathieu Karamitros, Sylvain Meylan, Vaclav Stepan,
Yann Perrot, Sébastien Incerti

G4Molecule & co

G4MoleculeDefinition **G4MolecularConfiguration**	Define an ID of a given molecule **G4MoleculeDefinition** defines **static and default features** (eg: number of atoms) **G4MolecularConfiguration** represents an ID for a given molecule configuration (electronic configuration, diffusion coefficient, charge). **G4MolecularConfiguration contains a pointer to G4MoleculeDefinition.**
G4Molecule	Inherits from **G4VUserTrackInformation**. Holds a pointer to **G4MolecularConfiguration**. **All G4Molecule of the same kind point to the same G4MolecularConfiguration.**
G4Track	G4Molecule can be accessed by: **GetMolecule**(track)

G4EmDNAChemistry

```
#include "G4VPhysicsConstructor.hh"
#include "G4VUserChemistryList.hh"
#include "globals.hh"
class G4DNAMolecularReactionTable;
class G4EmDNAChemistry : public G4VUserChemistryList
{
public :
        G4EmDNAChemistry();
        virtual ~G4EmDNAChemistry();
        virtual void ConstructMolecule();
        virtual void ConstructProcess();
        virtual void ConstructDissociationChannels();
        virtual void ConstructReactionTable(G4DNAMolecularReactionTable* reactionTable);
        virtual void ConstructTimeStepModel(G4DNAMolecularReactionTable* reactionTable);
};
```

```
void G4EmDNAChemistry::ConstructMolecule() {
  //---------------------------------
  // Create the definition
  G4H2O::Definition();
  G4Hydrogen::Definition();
  G4H3O::Definition();
  G4OH::Definition();
  G4Electron_aq::Definition();
  G4H2O2::Definition();
  G4H2::Definition();
  //---------------------------------
 // Define molecule model
  G4MoleculeTable::Instance()->CreateMoleculeModel("H3Op", G4H3O::Definition());
  G4Molecule* OHm = G4MoleculeTable::Instance()->   CreateMoleculeModel(
                                        "OHm", // just a tag to store and retrieve from G4MoleculeTable
                                        G4OH::Definition(),
                                       1, 5.0e-9 * (m2 / s)); // define new diffusion coefficient
  OHm->SetMass(17.0079 * g / Avogadro * c_squared);  // Set a new mass
  G4MoleculeTable::Instance()->CreateMoleculeModel("OH", G4OH::Definition());
  G4MoleculeTable::Instance()->CreateMoleculeModel("e_aq",
                                 G4Electron_aq::Definition());
  G4MoleculeTable::Instance()->CreateMoleculeModel("H",
                                 G4Hydrogen::Definition());
  G4MoleculeTable::Instance()->CreateMoleculeModel("H2", G4H2::Definition());
}
```

Useful commands

Create a new molecule

G4MoleculeDefintion* myMoleculeDefinition = **G4MoleculeTable::Instance**()->**CreateMoleculeDefinition**(

"NewMolDef", // name of the new molecule

5.0e-9 * (m2 / s)); // default diffusion coefficient

// WARNING: the definition of diffusion coefficient will be moved to processes

Create a "molecule model"

G4MoleculeTable::Instance()->CreateMoleculeModel(

"myMoleculeModel", // name of the new molecule

myMoleculeDefinition,

-1, // FACULTATIVE: select a model with a given charge

3.0e-8 * (m2 / s));

// FACULTATIVE: define a diffusion coefficient for the molecule of the selected charge

// WARNING: in the future, the definition of diffusion coefficient will be moved to processes and will handle multiple materials

Add reactions

```
void G4EmDNAChemistry::ConstructReactionTable(G4DNAMolecularReactionTable*
                        theReactionTable)
{
 G4Molecule* OHm = G4MoleculeTable::Instance()->GetMoleculeModel("OHm");
 G4Molecule* OH = G4MoleculeTable::Instance()->GetMoleculeModel("OH");
 G4Molecule* e_aq = G4MoleculeTable::Instance()->GetMoleculeModel("e_aq");
 G4Molecule* H2 = G4MoleculeTable::Instance()->GetMoleculeModel("H2");

 //------------------------------------------------------------
 // e_aq + e_aq + 2H2O -> H2 + 2OH-
 G4DNAMolecularReactionData* reactionData = new G4DNAMolecularReactionData(
       0.5e10 * (1e-3 * m3 / (mole * s)),  // reaction rate constant
       e_aq,   // reactant 1
        e_aq); // reactant 2
 reactionData->AddProduct(OHm);  // product 1
 reactionData->AddProduct(OHm); // product 2
 reactionData->AddProduct(H2);     // product 3
 theReactionTable->SetReaction(reactionData);  // add to the reaction table

...
```

chem1

Initializing & starting the chemistry

chem1: classes

chem1.cc	Main file ⇒ creation of runManager; actionInitialization; detectorConstruction; physicsList
→ **ActionInitialization**	**Build()** ⇒ creation of primaryGeneratorAction & **stackingAction**
→ **DetectorConstruction**	Definition of the geometry
→ **PhysicsList**	Choice of the physics & chemistry lists ⇒ G4EmDNAPhysics & **G4EmDNAChemistry**
→ **PrimaryGeneratorAction**	Choice of the primary particle
→ **StackingAction**	In simple words, in Geant4, when no more “physical tracks” remain, the method **StackingAction::NewStage** is called

chem1: Activate chemistry

Minimum commands to start ...

File	Method/function	Commands

chem1.cc	**main()**	//Activate the G4DNAChemistryManager
		G4DNAChemistryManager::Instance()
		->SetChemistryActivation(true);
		
		//Initialization
		G4DNAChemistryManager::Instance()-
		>InitializeMaster();

src/PhysicsList.cc	**PhysicsList::** **PhysicsList()**	**G4DNAChemistryManager::Instance()** **->SetChemistryList(new** **G4EmDNAChemistry());**
src/StackingAction.cc	**StackingAction::** **NewStage()**	//Start handling of chemistry tracks **G4DNAChemistryManager::Instance()->Run();**

chem1: usage of macros

/IT/verbose 1

#up to 4

/IT/endTime 1 us

stop at 1 microsecond

Or alternatively

#/IT/maxStepsNumber 10

stop after 10 steps

/IT/maxNullTimeSteps 10

stop after 10 null time steps
#(computed time steps = 0)

All macro commands can be browsed through the Qt interface, just do:

$ **chem1 -gui**

On the left hand side

⇒ macros with explanations

To obtain all possible command line options:

$ **chem1 -h**

chem2

Time step whatever you want

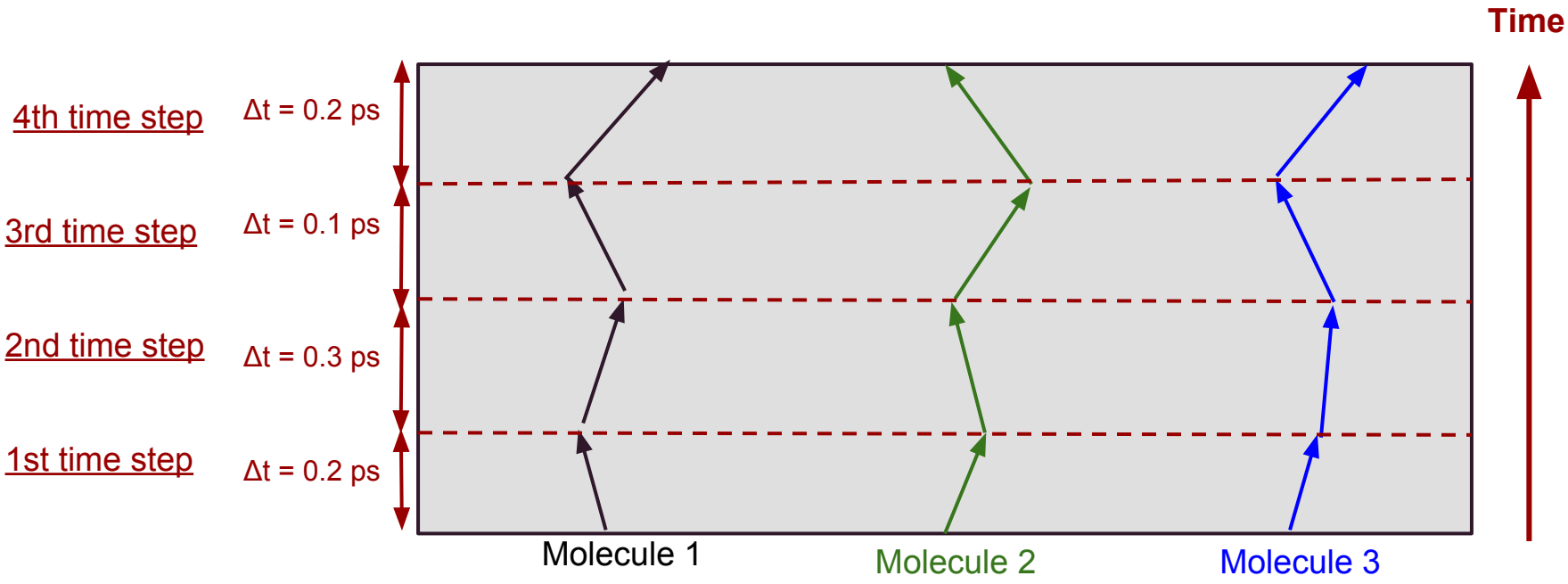
chem2: classes

chem2.cc	Main file ⇒ creation of runManager; actionInitialization; detectorConstruction; physicsList
→ **ActionInitialization**	**Extended to include TimeStepAction**
→ **DetectorConstruction**	Definition of the geometry
→ **PhysicsList**	Choice of the physics & chemistry lists ⇒ G4EmDNAPhysics & **G4EmDNAChemistry**
→ **PrimaryGeneratorAction**	Choice of the primary particle
→ **StackingAction**	In simple words, in Geant4, when no more “physical tracks” remain, the method **StackingAction::NewStage** is called
→ **TimeStepAction**	**Allow the user to set minimal time step values and to retrieve informations from a given time step (molecule names, reaction products, etc…).**

chem2: TimeStep definition

Time step: **An amount of time when molecules will all move (and maybe react).**

Dynamically decided thanks to the implemented MolecularStepByStepModel.



chem2: classes

chem2.cc	Main file ⇒ creation of runManager; actionInitialization; detectorConstruction; physicsList
→ **ActionInitialization**	**Extended to include TimeStepAction**
→ **DetectorConstruction**	Definition of the geometry
→ **PhysicsList**	Choice of the physics & chemistry lists ⇒ G4EmDNAPhysics & **G4EmDNAChemistry**
→ **PrimaryGeneratorAction**	Choice of the primary particle
→ **StackingAction**	In simple words, in Geant4, when no more “physical tracks” remain, the method **StackingAction::NewStage** is called
→ **TimeStepAction**	**Allow the user to set minimal time step values and to retrieve informations from a given time step (molecule names, reaction products, etc…).**

chem2: Class methods

src/TimeStepAction.cc

In ActionInitilialisation: G4VScheduler::Instance()->SetUserAction(new TimeStepAction());

Methods

- **TimeStepAction():** Constructor of the TimeStepAction class. Inside it you can set the minimal time steps of your simulation.

AddTimeStep(1 * picosecond, 0.1 * picosecond);	*During the first simulated picosecond the minimal time step will be of 0.1 picosecond. If molecules are too close and can react before that time limit: brownian bridge.*
AddTimeStep(10 * picosecond, 1 * picosecond);	*From 1 ps to 10 ps in simulation time, the minimal time step will be of 1 ps.*

chem2: Class methods

src/TimeStepAction.cc

Methods

- **Usual destructor and copy methods:** TimeStepAction(), ~TimeStepAction(), operator=().

- **StartProcessing()**: Beginning of the chemistry simulation.
- **EndProcessing()**: End of the chemistry simulation.

- **UserPreTimeStepAction()**: If the user want to do something before the start of the current time step.
- **UserPostTimeStepAction()**: If the user want to do something after the end of the current time step.
- **UserReactionAction(Reactif1, Reactif2, Products)**: will be called just after a reaction happened.

Since all those methods are empty the output you will see come from the ActionInitialisation verbose setting:
G4VScheduler::Instance()->SetVerbose(1);

chem2: Print the time spent during the last time step

src/TimeStepAction.cc

```
#include "G4ITScheduler.hh"

void TimeStepAction::UserPostTimeStepAction()
{
  G4cout<<G4ITScheduler::Instance()->GetTimeStep()<<G4endl;
}
```

G4ITScheduler::Instance() give access to many interesting methods:

- GetGlobalTime()
- GetEndTime()
- EndTracking()
- GetNTracks()
- GetMaxNbSteps()
- GetStatus()
- GetLimitingTimeStep() ...

chem3

User interactivity and visualization

Extensions to chem2

chem3.cc	Main file ⇒ creation of runManager; actionInitialization; detectorConstruction; physicsList
→ **ActionInitialization**	Extended to include ITTrackingInteractivity()
→ **DetectorConstruction**	World as semitransparent sphere now
→ **ITSteppingAction**	To retrieve information from a given “step” (several steps in one time step). Similar to SteppingAction but for chemistry.
→ **ITTrackingAction**	To retrieve information from a given track. Similar to TrackingAction but for chemistry.
→ **ITTrackingInteractivity**	Mandatory to visualize chemistry tracks.

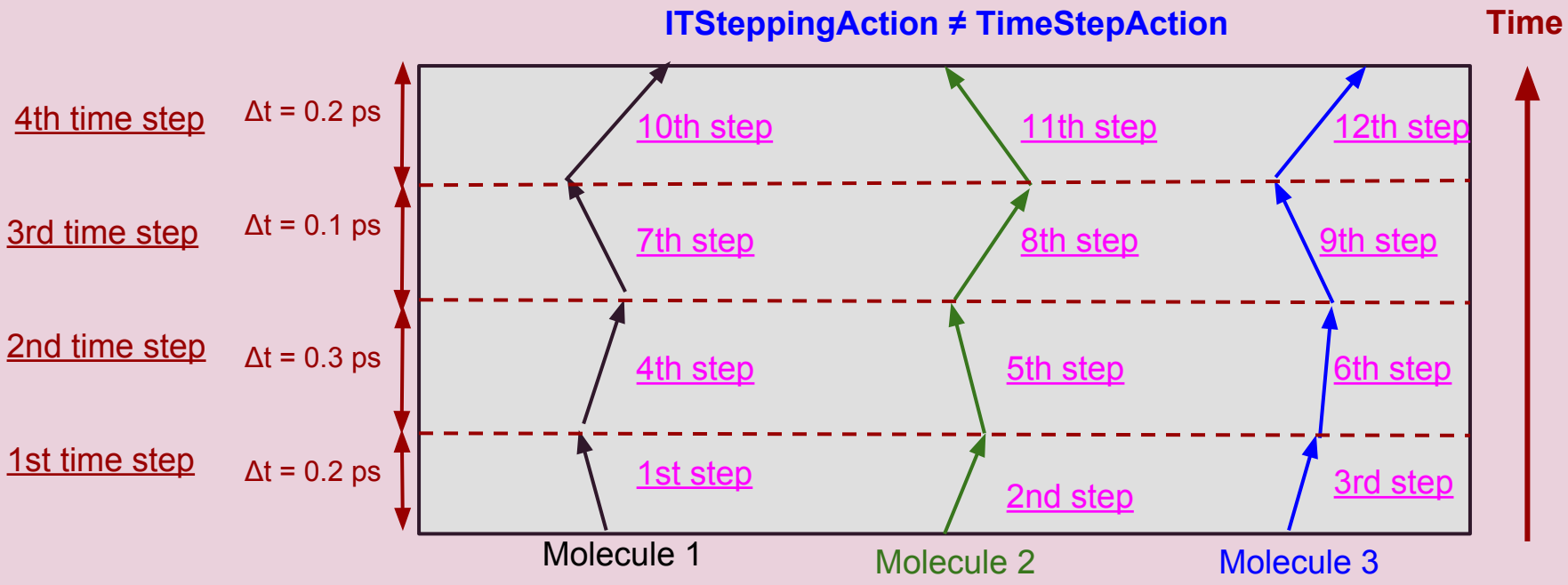
chem3: Class methods

src/ITSteppingAction.cc

In ActionInitilialisation: G4VScheduler::Instance()->SetUserAction(new ITSteppingAction());

Methods

- **ITSteppingAction() and ~ITSteppingAction:** Empty constructor and destructor.
- **UserSteppingAction()**: Method called after each step.



chem3: Class methods

src/ITTrackingAction.cc

In ActionInitilialisation: G4VScheduler::Instance()->SetUserAction(new ITTrackingAction());

Methods

- **ITTrackingAction() and ~ITTrackingAction():** Empty constructor and destructor.
- **PreUserTrackingAction(G4Track*):** Call at the start of a molecular track.
- **PostUserTrackingAction(G4Track*):** Call after the end of a molecular track

Note the useful method: **GetMolecule(track)->GetName()**
It can be used in pre or post user actions.

chem3: Class methods

src/ITTrackingInteractivity.cc

Methods

The methods here should not be changed by the user.

Executing the example

```
$ ./chem3 -gui

# Send one primary particle in
/control/execute beam.in
/vis/viewer/refresh

# Let's try a longer track
/gun/particle proton
/gun/energy 10 MeV
/run/beamOn 1
```

Using OpenGL for track visualization

#vis.mac: Enabling the time slices, see Drawing by time

/vis/modeling/trajectories/drawByParticleID-0/default/setTimeSliceInterval 0.1 ns

Manually setting the start and end time of tracks to show

/vis/ogl/set/startTime 0 s

/vis/ogl/set/endTime 1 ps

Filtering the track works as on normal tracks

/vis/filtering/trajectories/create/particleFilter

/vis/filtering/trajectories/particleFilter-0/add e-

/vis/filtering/trajectories/particleFilter-0/add e_aq

Please refer to excellent guides by Joseph Perl for more information.

vis.mac

```
# To show many tracks, but consume more memory
/vis/ogl/set/displayListLimit 10000000

# To draw chemical molecule trajectories
/vis/modeling/trajectories/create/drawByParticleID
/vis/modeling/trajectories/drawByParticleID-0/default/setDrawStepPts false
/vis/modeling/trajectories/drawByParticleID-0/default/setStepPtsSize 1

# ...in brave colors
/vis/modeling/trajectories/drawByParticleID-0/set OH magenta
/vis/modeling/trajectories/drawByParticleID-0/set H3O yellow
/vis/modeling/trajectories/drawByParticleID-0/set e_aq blue
/vis/modeling/trajectories/drawByParticleID-0/set H2O2 green
/vis/modeling/trajectories/drawByParticleID-0/set H white
```

Please refer to excellent guides by Joseph Perl for more information.

Scripting time steps visualization and camera movements using loops

physics.mac

```
/control/alias endTime 0.01

/vis/modeling/trajectories/drawByParticleID-0/default/setDrawStepPts true

/vis/ogl/set/fade 0

/vis/ogl/set/startTime 0 ps
/control/divide timeStep {endTime} 100.
/control/loop physics.loop startTime 0.0 {endTime} {timeStep}
```

physics.loop:

```
/vis/ogl/set/endTime {startTime} ps
```

Please refer to excellent guides by Joseph Perl for more information.

TimeSlices are memory hungry

Options:

- ➔ Avoid drawing step points (setDrawStepPts false)
- ➔ Limit chemistry simulation to shorter time

Safeguards:

- ➔ Lower displayListLimit
- ➔ Use bash ulimit command ([NNN] = kB)

```
ulimit -v 4194304
```

Additional

chem2: Safely retrieve the reaction products

src/TimeStepAction.cc

```
void TimeStepAction::UserReactionAction(const G4Track& a, const G4Track& b,
                        const std::vector<G4Track*>& products)
{
  // Check that if the address is pointing to something or not.
  if(&products!=0)
  {
    // If it pointing to a std::vector<G4Track*> then loop on all the G4Track*.
    // Each G4Track* points to one product.
    for(unsigned int i=0;i<products.size();++i)
    {
      G4Track* product = products[i];

      // to get the product name
      G4String productName = product[i]->GetDynamicParticle()->GetDefinition()->GetParticleName();

      // Do whatever you want

    }
  }
}
```